

UNITED STATES PATENT APPLICATION

OF:

HIANG-SWEE CHIANG

FOR:

WEB APPLICATION GENERATOR

Morgan & Finnegan, L.L.P.
345 Park Avenue
New York, NY 10154-0053
(212) 758-4800
Facsimile: (212) 751-6849
www.morganfinnegan.com

CROSS-REFERENCE TO RELATED APPLICATION

This application claims priority from provisional United States Patent Application Serial No. 60/190,364 entitled HTML TO WEB APPLICATION GENERATOR filed on March 17, 2000, the entirety of which is incorporated by
5 reference herein.

FIELD

A method, system and apparatus for generating web application source code, and, more specifically, a method, system and apparatus that reads a set of graphical web application screens as an input and generates the basis for a complete
10 web application source code as the output.

BACKGROUND

The increasing popularity of the World Wide Web and other Internet-based applications has encouraged many software companies and corporations to allocate major resources towards web-enabling existing or new applications. For example, most
15 retail applications conducted solely by phone or face-to-face in the past are now also conducted electronically over the World Wide Web.

Several approaches are used by web application developers to develop web applications. Each web application generally consists of the application framework, the application graphical user interface (GUI), application event handler(s) and the
20 application business logic. A first approach initiates with graphic designers and/or business analysts specifying web application screens in Hypertext Markup Language (HTML) format using visual HTML editors such as Microsoft FrontPage, Netscape

Composer and Adobe GoLive. The resulting HTML files are then given to web developers who specialize in computer programming. The web developers choose a traditional programming language such as C, C++, or Perl, prepare the programming source code files and embed the HTML tags from the HTML files provided by the graphic designers/business analysts directly into the web application programming source code. The web developers must also write the application framework and event handler code. All application source code files are then compiled into executable objects that are later installed within a web application server for access by web browsers.

However, after the web developers embed the HTML tags into the application source code, it is difficult for the graphic designers/business analysts to visualize how the web application screens appear by analyzing the HTML tags intermixed with the programming language code. As a result, modifying or changing the graphical user interface is difficult after the initial development. Further, each graphical user interface modification requires the web developers to re-compile the application source code and reinstall the object code within the web application server. Finally, the web developers must manually write the application framework and event handler code, which is a complex task and often introduces errors into the web application.

A second approach also initiates with graphic designers and/or business analysts specifying web application screens in HTML format using a visual HTML editor. The resulting HTML files are then given to web developers who specialize in computer programming. However, instead of using a traditional programming language such as C, C++ or Perl to write the web application, the web developers write scripting language

code such as JavaScript, VBScript, PHP Hypertext Preprocessor (PHP) or ColdFusion directly into the HTML files. All application code files are later interpreted at runtime based on a web request.

However, because of the scripting nature of the programming language, it is difficult for web developers to write structural or object-oriented code. Also, the final application source code includes many different HTML files whose structure and naming system do not reflect the purpose of the code residing inside them. Further, it is difficult for web developers to add functionality using this approach. Finally, since most scripting languages are interpreted at runtime based on a web request, the runtime performance of the web application is much slower than for a compiled web application.

A third approach, the Enhydra XMLC compiler introduced by Lutris Technologies, initiates with graphic designers and/or business analysts specifying web application screens in either HTML, Website Markup Language (WML) or Extensible Markup Language (XML) format. The resulting files are then converted by the XMLC compiler into a Java class containing an industry standard Document Object Model (DOM) tree. This DOM tree is a collection of Java objects that allows web developers to remove, update or add new content to the DOM tree. Also, this conversion separates the input files into an XML class that the graphic designers/business analysts can manipulate, separate from the web developers preparing the application framework and business logic. Thus, graphic designers/business analysts can view and test the application screens as a standalone component, while web developers prepare the application source code.

However, the Enhydra XMLC compiler does not generate application framework code or event handler code, nor does it generate any type of skeleton for the application framework or event handler methods to assist the web developers. In fact, the DOM object tree requires further manipulation by the web developers, including
5 setting all attributes, to convert the DOM object tree into a "string" which is readable by a web browser within the Transmission Control Protocol/Internet Protocol (TCP/IP). Thus, the Enhydra XMLC Compiler approach relies on the web developers to write the application framework code, event handler code, and develop the code for displaying outputs using the generated DOM object tree. Further, after the initial files created by
10 the graphic designers/business analysts are compiled by the XMLC compiler, any changes to the web application screens by the graphic designers/business analysts must be manually re-compiled by the web developers before the web application is available based on a web browser request.

SUMMARY

15 The above identified problems are solved and a technical advance is achieved by the web application generator. In accordance with one embodiment of the web application generator, there is provided a method of generating computer code for a web application, and dynamically binding input files from graphic designers and source code from web developers, comprising the web application server receiving
20 input files from graphic designers or business analysts, wherein the input files are at least one web application graphical user interface. The web application server determines if an application framework code is available for the web application, and retrieves the application framework code from an application directory. If the application

framework code is not available for the web application, then the web application server generates the application framework code, along with a business logic foundation code, an event handler skeleton and a graphical user interface code. Further, based on work by web developers, the web application server receives web application business logic objects and event handlers from the web developers, and organizes the application framework code, web application business logic objects and event handler methods into web application source code. The method further comprises compiling the web application source code. Modified input files may then be received by the web application server from the graphic designers or business analysts, and the modified input files are compiled and dynamically bound with the compiled web application source code at runtime. Generating the event handler skeleton comprises parsing the input files, reviewing each parsed input file for a tag type, attribute name and attribute value, and determining an event handler method based on the tag type, attribute name and attribute value.

An advantage of the web application generator is that the graphic designers can modify the web application graphical user interface at the same time that the web developers are preparing the web application business logic objects and event handler methods. Also, modifications to the input files that make up the graphical user interface are automatically re-compiled or re-interpreted at runtime without affecting the parallel work of the web developers. Further, the event handler skeleton, business logic foundation code and graphical user interface code provide web developers with a framework for finalizing the web application source code.

It is not intended that the method, system and apparatus for providing a web application generator be summarized here in its entirety. Rather, further features, aspects and advantages of the web application generator are set forth in or are apparent from the following drawings, detailed description and claims which follow.

5

BRIEF DESCRIPTION OF THE DRAWINGS

Further aspects of the method, system and apparatus for providing a web application generator will be more readily appreciated upon review of the detailed description set forth below when taken in conjunction with the accompanying drawings, of which:

10

FIG. 1 is a network schematic of an embodiment of the web application generator system by which web applications are accessed;

FIG. 2 is a network schematic of an embodiment of the web application generator system by which web applications are generated;

15

FIG. 3 is a block diagram of an embodiment of the web application server 100;

FIG. 4 is a block diagram of an embodiment illustrating the components of a web application 400;

FIG. 5 is a flow diagram 500 illustrating an embodiment of the process by which a web application is generated and sent to a web browser;

20

FIG. 6 is a flow diagram 600 illustrating an embodiment of the process by which input files are applied to the web application generator and web application source code is created;

FIG. 7 is a flow diagram 700 illustrating an embodiment of the process of applying input files to the web application generator;

FIG. 8 is an embodiment of web page 800 generated by HTML files:

FIG. 9 is an embodiment of web page 900 generated by HTML files; and

5 FIG. 10 is an embodiment of web page 1000 generated by HTML files and web application source code.

It will be understood that the foregoing brief description and the following detailed description are exemplary and explanatory of the method, system and apparatus for providing a web application generator, but are not intended to be
10 restrictive thereof or limiting of the advantages which can be achieved by the web application generator. Thus, the accompanying drawings, referred to herein and constituting a part hereof, illustrate embodiments of the web application generator and, together with the detailed description, serve to explain the principles of the web application generator.

DETAILED DESCRIPTION

The web application generator system is illustrated generally in FIG. 1.

First, users may request and access web applications in a variety of ways. For example, a user may access the Internet with a personal computer (PC) 105. The PC 105 usually contains web browser software, such as Microsoft Internet Explorer or Netscape Navigator. The user provides a uniform resource locator (URL) address using the web browser, which accesses the Internet in hypertext transfer protocol (HTTP) format to contact the application server 100 identified by the URL address. Based on the particular URL address, the application server 100 prepares the appropriate files and delivers them to the requesting web browser by means of the Internet. A user may also access the Internet with a PC which is part of a local area network (LAN). The user's PC in a LAN, such as PC 120, accesses a web application in the same manner as PC 105, except that the URL address request and response must additionally travel within the LAN 115 and through the LAN server 110 prior to reaching the Internet. Finally, a user may access a web application with a wireless device, such as a personal digital assistant (PDA) 135 or a cellular phone. For example, the PDA 135 contains web browser software that accepts a URL address input from the user, and accesses the Internet in a wireless fashion. Based on the URL address, the request is sent to the appropriate application server, where the web application requested is prepared and sent back to the requesting PDA. Thus, the web application may be requested in any technologically feasible fashion, including in a wireless fashion.

The web application generator/web application server is illustrated generally in FIG. 2. The web application server 100 comprises a web application

generator 205. Web applications are generally prepared by graphic designers/business analysts 210 and web developers 215. In one embodiment, the graphic designers/business analysts 210, on the one hand, create the web application screens (graphical user interface) in one of several available formats, such as XML and XSL (Extensible Style Language), HTML, cHTML (compact Hypertext Markup Language) and WML.

The choice of web application screen format depends on the types of devices targeted by the graphic designers/business analysts 210 and web developers 215 for accessing the web application being developed. For example, for web browser access only, the web application screen format can be HTML. For wireless application protocol (WAP) phone access only, the web application screen format can be WML. For i-mode phone access only, the web application screen format can be cHTML. Finally, for browser, WAP phone, i-mode phone and other device access, the web application screen format can be XML and XSL, particularly because XML is a "meta" language which is used to create other languages such as HTML, WML and cHTML. (XSL defines the standard stylesheet language for XML).

The web developers 215 are typically computer programmers who use a programming language, such as the Java language developed by Sun Microsystems, to write the underlying function for the web application.

Exemplary components of the web application server 100 are illustrated in FIG. 3. The primary component of web application server 100 is processor 305, which may be any commonly available microprocessor. Processor 305 may be operatively connected to further exemplary components, such as RAM/ROM 310, clock 315,

input/output devices 320 and memory 325 which, in turn, stores one or more computer programs 330 and databases 335.

Processor 305 operates in conjunction with random-access memory and read-only memory. The random-access memory (RAM) portion of RAM/ROM 310 may be a suitable number of Single In-Line Memory Module (SIMM) chips having a storage capacity (typically measured in kilobytes or megabytes) sufficient to store and transfer processing instructions utilized by processor 305 which may be received by application programs 330. The read-only memory (ROM) portion of RAM/ROM 310 may be any permanent non-rewritable memory medium capable of storing and transferring processing instructions performed by processor 305 during a start-up routine of web application server 100.

Clock 315 may be an on-board component of processor 305 which dictates a clock speed (typically measured in MHz) at which processor 305 performs and synchronizes communication between the internal components of web application server 100.

Input/output devices 320 may be one or more known devices used for receiving operator inputs, network data, and the like and transmitting outputs resulting therefrom. Accordingly, exemplary input devices may include a keyboard, a mouse, a voice recognition unit and the like for receiving operator inputs. Output devices may include any known devices used to transmit data, including transmitting data over the Internet. Other input/output devices may include a telephone or network connection device, such as a telephone modem, a cable modem, a T-1 connection, a digital

subscriber line or a network card, for communicating data to and from other computer devices over the Internet.

Memory 325 may be an internal or external large capacity device for storing computer processing instructions, computer-readable data, and the like. The

5 storage capacity of memory 325 is typically measured in megabytes or gigabytes.

Accordingly, memory 325 may be one or more of the following: a floppy disk in conjunction with a floppy disk drive, a hard disk drive, a CD-ROM disk and reader/writer, a DVD disk and reader/writer, a ZIP disk and a ZIP drive, and/or any other computer readable medium that may be encoded with processing instructions in a read-only or

10 read-write format.

Memory 325 may store a plurality of programs 330. Programs 330 may include web application generator 205. Memory 325 also includes databases 335 comprising multiple blocks of information relevant to the web application.

A web application 400, illustrated generally in FIG. 4, comprises

15 application graphical user interfaces (GUI) 405 (prepared by the graphic designers/business analysts 210), application framework 410, application event handler 415 and application business logic 420. Application framework 410 provides instruction to the operating system of the web application server 100, and determines/coordinates the execution of each function performed by the web application. The application

20 framework 410 also acts as a conduit between the web browser request (in the form of a URL address) and the interaction of the graphics (prepared by the graphic designers/business analysts 210) and function (written by the web developers 215). The application event handler 415 is equivalent to a "click method" object in an object-

oriented programming language for handling requests based on the GUI <input> tags.

The application event handler 415 coordinates between the application framework 410 and the application business logic 420. The application business logic 420 controls the function portion of the web application. Also, the application business logic 420 is

5 generally invoked by the application event handler 415.

Web Application Development

The web application development and deployment involves a series of steps illustrated in FIGS. 5-7. As shown in FIG. 5, the first step comprises one or multiple graphic designers and/or business analysts 210 creating web application screens using visual editors or a text editor (step 505). In one embodiment, the application screens are written in HTML format using a visual HTML editor, such as Microsoft FrontPage, Adobe GoLive or Netscape Composer.

10 Once the graphic designer(s) 210 are satisfied that the application screens contain all the necessary visual input and output elements for the web application, all the application screens are applied to the web application generator as input files (step 510). In one embodiment, the input files are in HTML format.

In other embodiments, the web application screens may be in other formats, such as XML, XSL, WML and cHTML, depending on the types of devices targeted for the web application. All such languages are created using XML, and as a result, their syntax is exactly the same. For example, a simple HTML file is as follows:

```
<html>
<body>
<p> Hello World. </p>
</body>
25 </html>
```

The same file in WML format is as follows:

```
<wml>
<card>
<p> Hello World. </p>
</card>
</wml>
```

Thus, both examples show essentially the same tags but in different languages.

As illustrated in FIG. 6, the web application generator 205 reads the set of web application screens as the input 605 and generates web application source code 610 as the output. This web application source code output 610 comprises web application framework code 615, event handler code 620, graphical user interface (GUI) code 625, business logic foundation code 630 and additional files 635. The web application generator 205 generates the same application framework code 615, event handler code 620, graphical user interface code 625, business logic foundation code 630 and additional files 635 regardless of the language format for the input files, because the input tags are interpreted in the same fashion for the different languages (e.g., XML, HTML, etc.).

FIG. 7 illustrates how the input files are applied to the web application generator. As shown in FIG. 7, web application generator 205 first determines if application framework code 615 is already available by searching for its existence in the application directory (step 705). Application framework code is specific for a particular web application. Once it is generated for the web application, it is stored in the application directory of the application web server 100. However, a separate web application requires a separate application framework code. Further, the same application provided by graphic designers/business analysts in multiple input formats

(e.g., HTML and WML) uses the same application framework code and does not require multiple generations of application framework code.

Web application generator 205 also determines if business logic foundation code 630 was previously generated for the web application. Business logic foundation code is also specific for a particular web application. In one embodiment, business logic foundation code 630 is available if application framework code 615 is available. However, in another embodiment, application framework code 615 may be available while business logic foundation code 630 is not. In this case, business logic foundation code 630 may be generated by the web application generator 205.

If application framework code 615 is not found in the application directory, web application generator 205 generates application framework code 615 and business logic foundation code 630 (step 710). Application framework code 615 is generated independent of the content of the input files 605, and provides the framework for the entire web application. Business logic foundation code 630 is also generated independent of the input files 605, whereas event handler code 620 and GUI code 625 depend on the content of the input files 605. If application framework code 615 and business logic foundation code 630 were created earlier for this web application, step 710 is skipped.

Next, web application generator 205 reads each input file 605 (step 715), and parses the input file to create an object representation of the file for subsequent manipulation (step 720). After parsing the input file, web application generator 205 analyzes what is in the parsed file and, in one embodiment, attempts to identify certain input tags (step 725). Such tags include, but are not limited to, the following:

comment tag	password tag	JLMShowView tag
include tag	text area tag	JLMShowParentView tag
variable tag	select tag	repeat tag
output tag	checkbox tag	if else tag
frame tag	radio tag	
anchor tag	hidden tag	
form tag	JLMClick tag	
text tag	JLVDefaultClick tag	

After identifying the input tag, web application generator 205 searches for specific attributes within the input tag, based on the input tag type. In one embodiment, certain tags have a single attribute, while other tags have multiple attributes. After identifying each attribute within the tag, web application generator 205 identifies an

- 5 attribute value associated with the attribute name. Ultimately, based on the input tag, attribute name(s) and associated attribute value(s), web application generator 205 relies on a particular "rule" (or fixed formula) within web application server memory 325 to generate event handler code 620 and GUI code 625 (step 730). Thus, the input tag, attribute name and attribute value determine the "rule" that web application generator
- 10 205 relies on to generate event handler code 620 and GUI code 625.

For example, in one embodiment, one tag of an input file in HTML format is "<input type=submit name=JLMClick value=Add>". Web application generator 205 first identifies the tag as an "input" tag, next identifies the attribute names as "name" and "value", and further identifies the associated attribute values as "JMLClick" and "Add".

- 15 Based on the input tag, attribute names and associated attribute values, web application generator 205 relies on a particular rule in web application server memory 325 to generate the following event handler method: "public void add (App app) throws Exception", as well as a GUI code in the form of a source file such as "indexClick.java." If one of the attribute values was something other than JLMClick, then web application

generator 205 would have relied on a separate rule in web application server memory 325 to generate event handler code 620 and GUI code 625.

Alternatively, in another embodiment, one tag of an input file in WML format is:

5 <go href="./?JLMClick=Add"/>

Web application generator 205 identifies the input tag ("go"), attribute name ("href") and associated attribute value ("./?JLMClick=Add/"). Since this tag is the same input tag as the first example except in a different format (WML vs. HTML), web application generator 205 relies on the same rule as the first example to generate the same event handler method and GUI code.

In one embodiment, web application source code output 610 is generated as a corresponding Java class. However, in other embodiments, web application source code output 610 may be structured for any object-oriented programming language, such as JavaScript, C#, C++ or SmallTalk.

As illustrated in FIG. 6, web application source code output 610 comprises web application framework code 615, event handler code 620, graphical user interface (GUI) code 625, business logic foundation code 630 and additional files 635. In one embodiment, each portion of web application source code output 610 is generated in Java as follows based on HTML input files. In other embodiments, different input format languages and different programming languages may be used.

First, web application framework code 615 comprises an application object and a complete Java servlet web application framework object. In other embodiments, the servlet can be based on different object-oriented programming languages.

The application object is passed to all event handlers (click methods).

Also, the application object allows the event handlers (click methods) to access: (1) the HTTP request object received from the web browser (in the form of a URL address), (2) the HTTP response object sent back to the requesting web browser, (3) application business logic, (4) the application database connection, and (5) application structured query language (SQL) statements and application properties. Further, the application object allows event handlers (click methods) to set the dynamic values for <input> and <output> tags within an HTML screen, and allows event handlers (click methods) to show an HTML screen. Finally, the application object is immediately ready for use within the web application and requires no additions or revisions by web developers.

The Java servlet web application framework object receives requests from web browsers, dispatches those requests with the application object to appropriate event handlers (click methods), receives responses back from event handlers (click methods) via the application object and transmit responses back to requesting web browsers. The Java servlet web application framework object also reads the application properties file and makes all application properties available in the application object, and reads the application SQL file and makes all application SQL statements available in the application object. Further, the Java servlet web application framework object manages database connections for the application by establishing, sharing and terminating database connections, regardless of database types and application specific contents. Database connections are network connections from the web application server to the database specified by the web developers, such as an Oracle 8i database, Microsoft SQL server, etc. Finally, the Java servlet web application framework object is

immediately ready for use within the web application and requires no additions or revisions by web developers.

Event handler code 620 comprises a skeleton or framework made up of an event handler (click method) object that accepts requests from HTML <input> tags with JLMClick name and submit type attributes in the HTML files. As discussed above in connection with FIG. 7, for each JLMClick submit <input> tag identified by web application generator 205, a corresponding event handler (click method) is generated to handle the request. Upon receipt of event handler code 620 as an output from web application generator 205, web developers 215 simply need to write appropriate source code into the event handler code skeleton to drive the application business logic in response to user interactions.

Graphical user interface code 625 is a programming proxy to input files 605. Input files 605 contain output parameters that must be set up by web developers 215 as a result of the function, and web developers 215 need a way to set those parameters in an object-oriented programming environment. Graphical user interface code 625 provides this in the form of a programming proxy or view. Later, during the deployment phase, application framework code 615 binds the output parameters to the interpreted or compiled input files, developer by graphic designers/business analysts 210.

Next, business logic foundation code 630 is the foundation for the application business logic object that is fully prepared by web developers 215 and accessible to all event handlers (click methods) by means of the application object. Upon receipt of business logic foundation code 630 as an output from web application

generator 205, web developers 215 assemble reusable object-oriented business components and write all the core business logic that drives the web application, and particularly its function.

Web developers 215 may ignore business logic foundation code 630, and
5 prepare the application business logic objects from "scratch." However, business logic foundation code 630 is provided to assist web developers 215 with a foundation or skeleton, but may be completely ignored by web developers 215. In fact, in another embodiment, no business logic foundation code is generated whatsoever as part of the output from the web application generator.

10 Finally, additional files 635 are generated by web application generator 205, allowing information to be initially set by web developers 215, but changeable by application administrators without requiring a recompilation of the application source code. For example, in one embodiment, an application runtime properties file is generated where web developers set all the application runtime properties which are
15 made available to event handlers (click methods) by the application servlet framework. This application runtime properties file may be changed by application administrators during deployment by simply editing the file rather than editing the application source code and re-compiling the application source code. Also, an application SQL statements file may also be generated where web developers set all the application
20 SQL statements which are made available to event handlers (click methods) by the application servlet framework. This application SQL statements file may be changed by application administrators during deployment on different databases by simply editing

the file rather than editing the application source code and re-compiling the application source code.

Returning to FIG. 5, web developers 215 prepare web application source code in an object-oriented programming language based on event handler code 620, graphical user interface code 625 and business logic foundation code 630 (step 520). As described in detail above in connection with FIG. 6, the web developers 215: (1) write appropriate source code into event handler code 620 to drive the application business logic in response to user interactions, (2) refer to graphical user interface code 625 as a programming proxy, and (3) optionally rely on business logic foundation code 630 to assemble reusable object-oriented business components and write all the core business logic that drives the web application.

As web developers 215 are busy adding functionality to event handler code 620 and business logic foundation code 630, graphic designers/business analysts 210 determine if the web application screens require further edits (step 515). If the graphic designers/business analysts 210 elect to edit the web application screens, they can work in parallel (with the web developers 215) on the application screens to alter and/or improve the design of the graphical user interface (step 525). Such design changes takes place immediately without requiring web application generator 205 to re-generate web application source code 610 or web developers 215 to modify and re-compile existing source code. However, at runtime, application framework code 615 detects any change to the web application screens, since such screens were initially inputs 605 to web application generator 205, and if changes are detected, the input files are either re-compiled or re-interpreted automatically. In one embodiment, the graphic

design is greatly simplified because the HTML files contain pure HTML tags without any confusing script. Thus, the HTML files can be easily manipulated by graphic designers/business analysts 210 using any visual HTML editor. Also, no HTML tag is embedded in the programming source code and therefore, the user interface is 100% customizable just by editing the HTML files without having to edit the source code.

Also, web developers 215 benefit from the separation of the HTML tags from the programming code because the web developers 215 can use any object-oriented programming language such as Java to develop the web application rather than using a scripting language that requires line-by-line interpretation. An object-oriented programming language allows the web application to achieve better scalability in terms of providing additional features and complexity in the complete life cycle of the web application. Also, the use of a compiled programming language (as opposed to an interpreted programming language) allows a better runtime performance for the web application than a web application developed with an interpreted scripting language.

When web developers 215 are finished preparing web application source code 610, web application source code 610 is either compiled or interpreted by the programming language compiler/interpreter in which the code is written (step 530). In one embodiment, the web application source code is written in the Java programming language and compiled by a Java compiler. In other embodiments, the web application source code may be written and compiled or interpreted in JavaScript, C#, C++, SmallTalk or other programming languages.

At some point, graphic designers 210 complete any design changes to the graphical user interface and web application source code 610 prepared by web

developers 215 is either compiled or interpreted. Thus, the development phase is complete and the web application is ready for deployment.

Web Application Deployment

Based on a request for the web application from a web browser, the web application server 100 dynamically binds graphical user interface (GUI) files with web application business logic objects at runtime in order to provide the web application to the web browser (step 535). As described above, if the web application screens were modified after initially being applied to web application generator 205, they are automatically re-compiled or re-interpreted by application framework code 615. In one embodiment, HTML, WML or cHTML input files can either be interpreted or compiled by the application framework at runtime. Compilation of the input files (instead of interpreting them) increases application framework performance as well as complexity. Further, the input files may be compiled at runtime into Java Document Object Model (DOM) objects and the resulting objects are stored in a cache in the computer memory for performance purposes. In another embodiment, XML and/or XSL files can also either be interpreted or compiled by the application framework at runtime. For example, at runtime, the application framework compiles the XML files into DOM objects, transforms the DOM objects into the appropriate DOM objects based on the XSL files, caches the transformed objects and binds the data from the business logic components to produce the final screen viewed by the application users.

Cost Calculator Example

In an exemplary embodiment, a cost calculator web application is developed as follows. First, a graphic designer/business analyst creates a cost

calculator screen using a visual HTML editor or a text editor to produce the following HTML tags within an HTML file:

```
1. <!-- File: CostCalculator.html -->
2. <html>
3. <head><title>Cost Calculator</title></head>
4. <body>
5. <h3>Cost Calculator</h3>
6. <form method=post action=../>
7. Item Code: <input type=text name=item_code><br>
8. Item Quantity: <input type=text name=item_qty><br>
9. Unit Price: <output name=unit_price></output><br>
10. Total: <output name=total_cost></output><br>
11. <input type=submit name=JLMClick value=Calculate>
12. </form>
13. </body>
14. </html>
```

Line 1 is an HTML comment showing the name of the HTML file. Lines 2-14 contain the HTML tags for the HTML document. Line 3 contains the "head" part of the HTML document, that sets the title of the document to "Cost Calculator." This title appears in the title bar of most web browsers receiving this web application. Lines 4-13 contain the "body" part of the HTML document. Line 5 shows a "Cost Calculator" heading at the top of the document. Lines 6-12 contain the "form" part of the HTML document, allowing users to send information to the web application. For example, line 7 produces a text box for a user to enter a product item code. Line 8 produces another text box for a user to enter the product item quantity. Lines 9 and 10 show the unit price and total cost for the product item. The <output> tag in lines 9-10 is processed by the web application server on the server-side while all the other HTML tags are processed by the web browser on the user-side. Finally, line 11 shows a "calculate" button for the user to click in order to send the item code and quantity to the web application server for

computation of the total cost. The initial cost calculator screen 800 is illustrated in FIG. 8.

After the graphic designer/business analyst is satisfied that the HTML screen contains all the necessary input and output elements for the cost calculator web application, the HTML files are sent through the web application generator in order to create the web application source code. The web application source code created comprises Java files that make up the application framework code, event handler code, graphical user interface code, business logic foundation code and additional application files. For example, the application framework code comprises an application object (App.java) and a complete Java servlet web application framework object (AppServlet.java). Both the application object and the complete Java servlet web application framework object are described in detail above in connection with FIG. 6, and are ready for use within the web application and require no additions or revisions by web developers. The event handler code and graphical user interface code, also described in detail in connection with FIG. 6, comprise one object file (CostCalculator.java). This object file requires web developers to write additional source code to drive the application business logic in response to user interactions. Next, the business logic foundation code, described in detail in connection with FIG. 6, comprises a foundation file (AppModel.java) for the application business logic object that is accessible to all event handlers (click methods) by means of the application object file. Web developers may ignore this file, or use it to assemble reusable object-oriented business components and write all the core business logic that drives the web application. Finally, two additional files (App.properties, App.sql) are created that allow

the web developers to set the application runtime properties and application SQL statements for the web application. After the development phase, application administrators may change either the application runtime properties or the application SQL statements by simply editing either file rather than editing the application source code and re-compiling the application source code.

A portion of the web application source code created by the web application generator for the cost calculator is shown as follows:

// File: AppModel.java

// original file generated by WebAppGen

```
public class AppModel {  
    // application business logic variables and methods here  
    public void showHome(App app) throws Exception {  
        // application home page code here  
        app.showView("CostCalculator.html");  
    }  
    AppModel(AppServlet as) throws Exception {  
        // application initialization here  
    }  
}
```

// File: CostCalculator.java

// original file generated by WebAppGen

```
public class CostCalculator {  
    public static void calculate(App app) throws Exception {  
        // click method code here  
        app.showView("CostCalculator.html");  
    }  
}
```

// File: AppModel.java

// added getUnitPrice business logic method

```
public class AppModel {  
    // application business logic variables and methods here
```

```

    public int getUnitPrice(String item_code) {
        // this is an example business logic method
        if (item_code.equals("20000412")) return 20;
        else return 50;
    }
    public void showHome(App app) throws Exception {
        // application home page code here
        app.showView("CostCalculator.html");
    }
    AppModel(AppServlet as) throws Exception {
        // application initialization here
    }
}

```

For example, Line 11 of the CostCalculator.html file ("") is an HTML tag that causes the web application generator to generate a corresponding "void calculate (App app)" event handler method based on the tag type, attribute name and attribute value. When a user clicks on the "Calculate" button within a web browser, the request to the web application is dispatched to the corresponding "calculate" method by the web application framework.

Next, the web developers add functionality to the AppModel.java and CostCalculator.java files created by the web application generator. Also, the web developers can set parameters, as shown below in the web application source code, to set the unit price and total cost values in the application object:

// File: CostCalculator.java

// added code to get the unit price
 // and compute the total cost
 // in the calculate click method

```

public class CostCalculator {
    public static void calculate(App app) throws Exception {
        // click method code here
        String item_code = app.getParameter("item_code");
        int unit_price = app.model.getUnitPrice(item_code);
        int item_qty = Integer.parseInt(app.getParameter("item_qty"));
        int total_cost = unit_price * item_qty;
    }
}

```

```

    app.setParameter("unit_price",String.valueOf(unit_price));
    app.setParameter("total_cost",String.valueOf(total_cost));
    app.showView("CostCalculator.html");
}
5  }
```

Later at runtime, the application uses these values to replace the <output name=unit_price></output> and <output name=total_cost></output> HTML tags in the CostCalculator.html file before sending as a response to the web browser.

The graphic designers may work in parallel while the web developers add
 10 functionality to the CostCalculator.html file to improve the design of the graphical user interface. For example, the graphic designers can modify the original CostCalculator.html file as follows:

```

1.  <!-- File: CostCalculator.html -->
2.  <!-- modified to make the user interface look better -->
3.  <html>
4.  <head><title>Cost Calculator</title></head>
5.  <body>
6.  <center>
7.  <h3>Cost Calculator</h3>
8.  <form method=post action=../>
9.  <table>
10. <tr><td>Item Code:</td>
11. <td><input type=text name=item_code></td></tr>
12. <tr><td>Item Quantity:</td>
13. <td><input type=text name=item_qty></td></tr>
14. <tr><td>Unit Price</td>
15. <td><output name=unit_price></output></td></tr>
16. <tr><td>Total</td>
17. <td><output name=total_cost></output></td></tr>
18. </table>
19. <input type=submit name=JLMClick value=Calculate>
20. </form>
21. </center>
22. </body>
23. </html>
35
```

Lines 6 and 21 were added to center the user interface within the browser.

Lines 9 and 18 were added to arrange the item code, item quantity, unit price and total

names and values in table format. Further, in lines 10-17, the <tr> and </tr> tags specify a row of the table while the <td> and </td> tags specify a cell of the table. The modified cost calculator screen 900 is illustrated in FIG. 9.

After the modifications are added to the CostCalculator.html file, the effect
5 takes place immediately without requiring the web application generator to re-generate the web application source code or the web developers to modify and recompile the existing source code. At runtime, the Cost Calculator.html files will be re-compiled automatically by the application framework.

Upon completion by the graphic designers of the HTML files and
10 compilation of the web application source code prepared by the web developers, the web application is available for use.

As shown in FIG. 10, the user inputs an item code and item quantity through a web browser. The web application takes those inputs and provides a unit price and total as outputs.

15 Although illustrative embodiments have been described herein in detail, it should be noted and understood that the descriptions have been provided for purposes of illustration only and that other variations both in form and detail can be made thereupon without departing from the spirit and scope of the method, system and apparatus for providing a web application generator. The terms and expressions have
20 been used as terms of description and not terms of limitation. There is no limitation to use the terms or expressions to exclude any equivalents of features shown and described or portions thereof, and the web application generator should be defined with the claims that follow.